SMT320

OSF Device Driver Specification

1. Introduction

This document defines the requirements for a Unix Device Driver for the SMT320 PCI card (otherwise known as the ADC-C40-PCI) on an Alpha/OSF UNIX workstation. Specifically AlphaStation 500/400, running Digital UNIX v4.0A.

2. Purpose

To enable use of the Sundance range SMT32x PCI cards within Alpha AXP workstations running the OSF operating system. The functionality will include but not be limited to booting user application code, providing server based communications to user applications running on the target system via the comm-port interface. Provide "mapping interfaces" to enable 'C40 initiator transfers.

3. Pre-requisites

The SMT320 will be supplied with the following:-

DOS based example code

Windows 95™ Device Driver, source and executables for host and TMS320C4x.

It will have been fully tested with software which exercise all necessary on-board and board interface functions, including interrupts.

An Alpha/OSF workstation will be supplied with complete ‘C’ development software and all manuals necessary to build a driver on the system.

A 3L ‘C’ Compiler will also be supplied to generate test code along with the necessary server sources from 3L.

4. Commercial details

4.1 Deliverables

Binary device driver for the Alpha/OSF system.

Source code of the device driver for the Alpha/OSF system with in-line comments.

Manual on usage of the device driver supplied on PC and OSF readable media detailing system requirements, Installation, use of driver, control of driver and simple troubleshooting guide.

Binary and source of the modified axp server program for the Alpha/OSF system based on previous version developed for Alpha Data Parallel Systems.

4.2 IPR/Copyright

Beam will retain the copyright on the device driver. Sundance will have permission to distribute, modify and sell the driver without any additional fees.

4.3 Maintenance

The deliverables will be supported for a period of 6 months from delivery with any necessary hardware/software systems supplied by Sundance.

4.4 Timing

Delivery will be effected as soon as practicable but within 4 weeks from receipt of the pre-requisites items. This work will be completed by the end of February 1997.

5. Technical specification

5.1 Name

SMT32X - SMT32x device driver

5.2 Description

The SMT32X device driver will be a character based driver, it provides read/write and control access to the SMT32x products used within the UNIX environment. It will be both loadable and static thus allowing end-user flexibility when building turnkey systems.

The SMT32X driver supports a single SMT32x PCI interface card. The device may be considered as a single function device however consideration should be given to the a possible future requirement for a multifunction

device where a system may be configured with SMT320 + SMT321 and SMT323.

The device driver will provide "mapping interfaces" to enable 'C40 initiator transfers to/from host memory or other devices.

The PCI BIOS in the Motherboard of system will set up the Address and Interrupt channel used by the board automatically.

Access to the SMT32x card is performed by a single special file entry:

/dev/SMT32X

Only one process may access the device at a time.

5.3 API Characteristics.

5.3.1 Block Transfers with C40 as Initiator.

The API will support C40 initiator transfers into/from HOST memory using pointers to Hardware (Physical addresses). These may need to be generated inside the Device Driver from virtual addresses passed by the application or Kernel. Initially, the Device Driver will maintain control until the transfer completes but may in future use overlapped I/O to improve performance.

Implementation of this is dependant upon receiving adequate code on the C40 and DOS/Windows host and information on how the board implements it.

5.3.2 Read/Write Target Control Registers (HOST Master)

The API will support per register transfers to/from configuration space. These functions will accept an offset into configuration space to select the register to be read/written. This function will support the target mode functions "Set C4X IIOF" and "Reset C4X" using a C40 control register in configuration space.

5.3.3 Block Transfer to Comm-port

The API function to perform this operation will take as its argument a memory block which will then be written to target space by Device Driver.

5.3.4 Interrupts.

The Device Driver will support PCI interrupts from the SMT32x.

5.4 Device Access

Normally the OSF process uses the device to provide access to the SMT32x system. However, in specialist applications the programmer can access the device using the following 'C' calls.

5.4.1 dev = open("/dev/SMT32X", 1)

Opens the default SMT32x device.

This function tries to open the device for read/write access. If the device cannot be opened then the call will return -1 and errno will be set to ENXIO. If the device is already in use by another process then the call will return -1 and errno will be set to EBUSY.

5.4.2 close(dev)

Closes the SMT32x device.

5.4.3 ioctl(dev, command, argument)

Sends control information to the driver.

See Section on ioctl commands for more information.

5.4.4 write(dev, buffer, nbytes)

Writes a sequence of bytes to the SMT32x system.

This function will write the required number of bytes from the given address to the SMT32x System. The function will return the actual number of bytes written or -1, with errno set to EIO, on fatal error.

5.4.5 read(dev, buffer, nbytes)

Reads a sequence of bytes from the SMT32x system. If the driver is in standard protocol mode then this function will read the required number of bytes from the SMT32x System to the given address. If the driver is in P5 protocol mode this function will read a complete P5 packet from the SMT32x System to the given address and return the number of bytes actually read.

Note: nbytes should be set to the maximum number of bytes to read in this case. The function will return the actual number of bytes read or -1, with errno set to EIO, on fatal error.

5.5 IOCTL Control Calls

A number of ioctl commands are supported, the basic 'C' syntax of an ioctl call is:

ioctl(file, command, argument);

The ioctl definitions are given in */usr/include/sys/SMT32X.h*. A list of the available commands follows.

5.5.1 [SMT32X_CRESET]

Reset the SMT32x system. No argument is required to the ioctl.

5.5.2 [SMT32X_CPROTOCOL]

Change the protocol to use for read operations. The Argument can be SMT32X_PR_NONE or SMT32X_PR_P5. See the read() function entry for more details.

5.5.3 [SMT32X_BTYPE]

Read the type of board in the system. Board types are:

SMT32X_BT_NONE	x	Unknown Board Type
SMT32X_BT_320	x	SMT320
SMT32X_BT_321	x	SMT321
SMT32X_BT_323	x	SMT323
SMT32X_BT_326	x	SMT326
SMT32X_BT_327	x	SMT327

This driver will return SMT32X _BT_320 with specific reference numbers to be defined by Beam to maintain compatibility with existing drivers.

NOTES On opening the device the SMT320 should be reset using the SMT32X_CRESET ioctl. After reset the driver expects the SMT320 kernel to be written to the driver in one chunk. Once the kernel has been written all further driver read/writes will be to the running SMT320 kernel. The SMT320 program to be run can now be written to the device.

FILES */dev/SMT32X*

SEE ALSO open(2), close(2), ioctl(2), read(2), write(2).